Stream Processing for High-Performance Embedded Systems

William J. Dally

Computer Systems Laboratory

Stanford University

HPEC

September 25, 2002

			Form Approved OMB No. 0704-0188
	Report Documentation Page		

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **25 SEP 2002**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**	
4. TITLE AND SUBTITLE **Stream Processing for High-Performance Embedded Systems**		5a. CONTRACT NUMBER	
		5b. GRANT NUMBER	
		5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)		5d. PROJECT NUMBER	
		5e. TASK NUMBER	
		5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Stanford University**		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)	
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release, distribution unlimited**			
13. SUPPLEMENTARY NOTES **Also see ADM001473 , The original document contains color images.**			
14. ABSTRACT			
15. SUBJECT TERMS			

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT **UU**	18. NUMBER OF PAGES **23**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Outline

- Embedded computing demands high arithmetic rates with low power

- VLSI technology can deliver this capability – but microprocessors cannot

- Stream processors realize the performance/power potential of VLSI while retaining flexibility
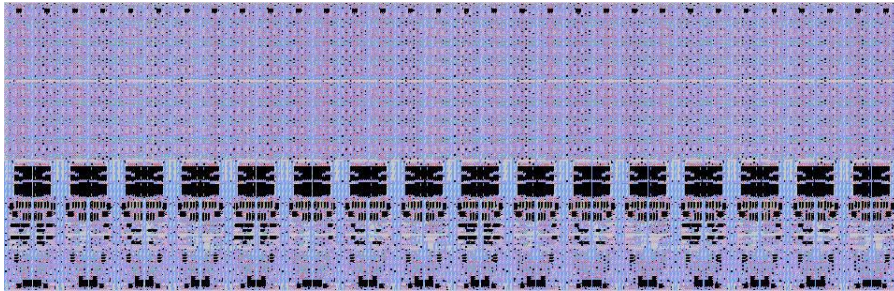
Embedded systems demand high arithmetic rates with low power
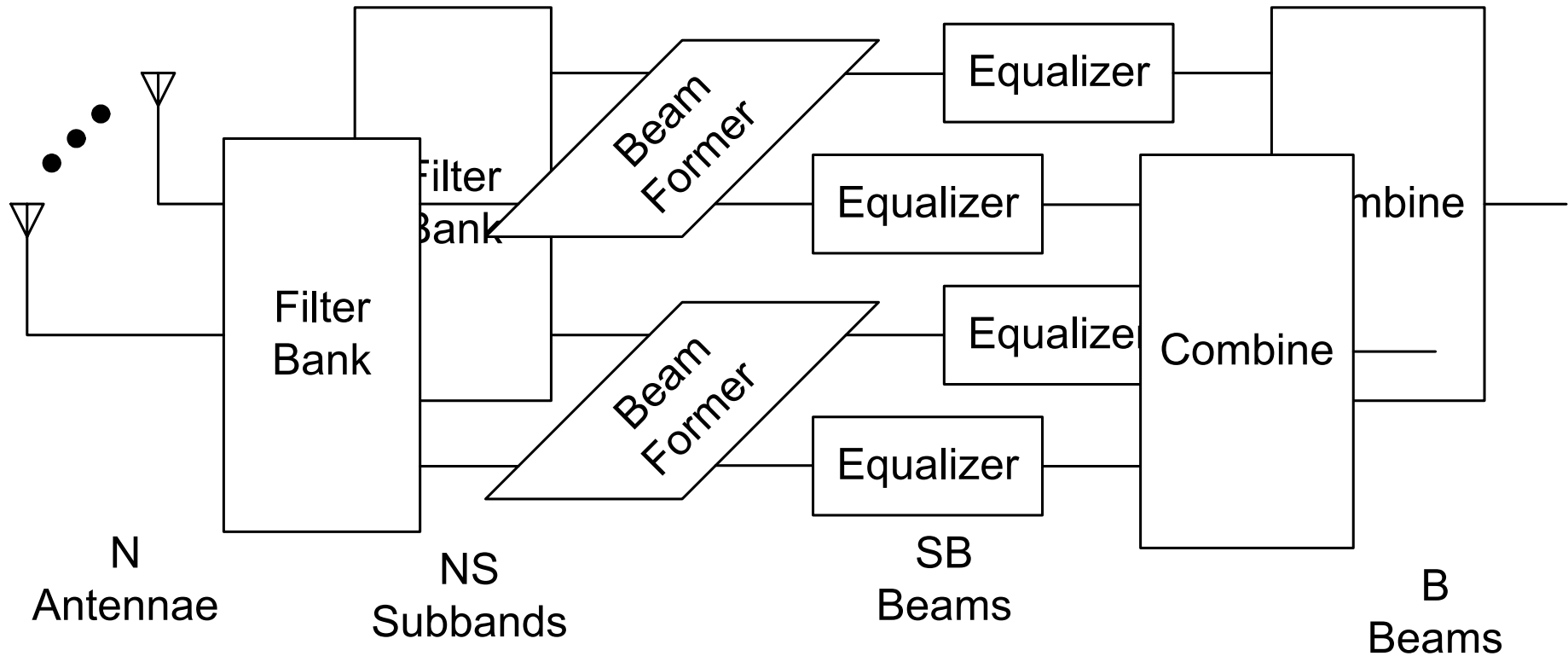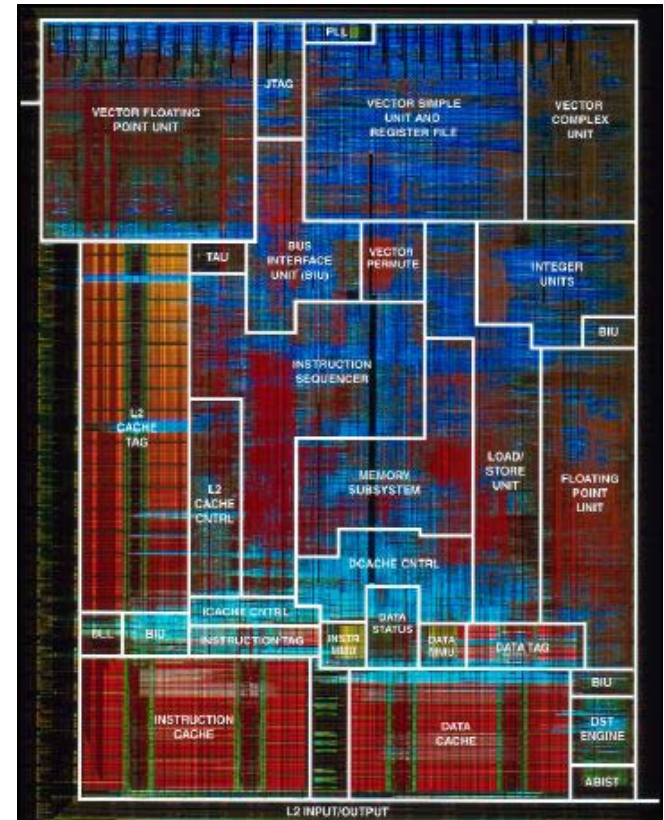

For N=10, BW=100MHz, S=16, B=4, about 500GOPs

VLSI provides high arithmetic rates with low power – microprocessors do not

PowerPC G4
$95mm^2$ ~1nJ/op

32b adder + RF, 512 x 163 tracks
$205\mu m$ x $65\mu m$ ~ $0.013mm^2$
~5pJ/op

Area 7300:1, Energy 200:1, Ops 4:1

# VLSI provides high arithmetic rates with low power – microprocessors do not

Operation	Energy (0.13um)	(0.05um)
32b ALU Operation	5pJ	0.3pJ
32b Register Read	10pJ	0.6pJ
Read 32b from 8KB RAM	50pJ	3pJ
Transfer 32b across chip (10mm)	100pJ	17pJ
Execute a uP instruction (SB-1)	1.1nJ	130pJ
Transfer 32b off chip (2.5G CML)	1.3nJ	400pJ
Transfer 32b off chip (200M HSTL)	1.9nJ	1.9nJ

300 : 20 : 1 off-chip to global to local ratio in 2002
1300 : 56 : 1 in 2010
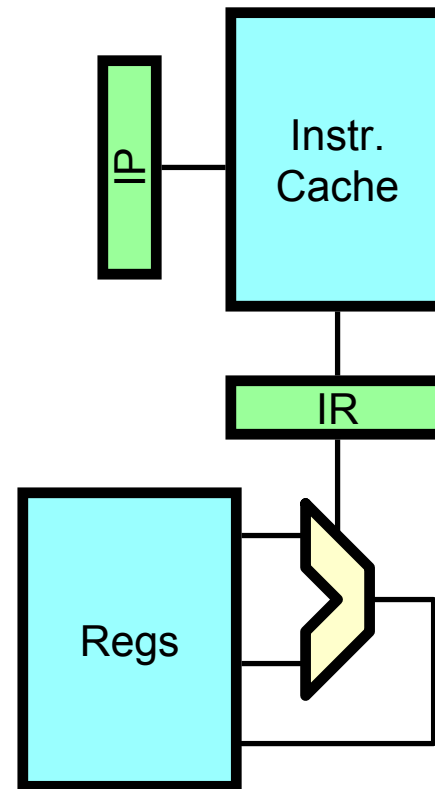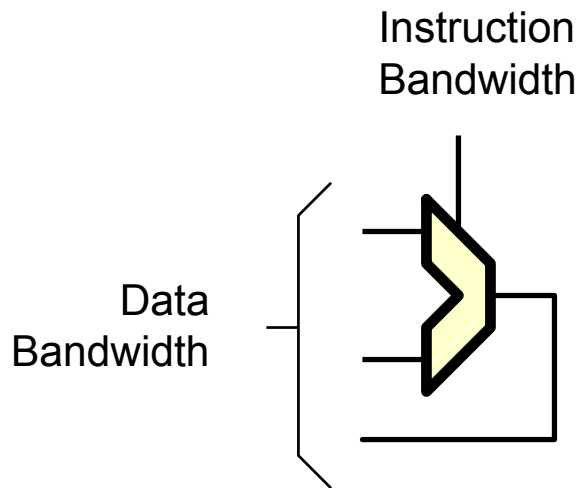
Why do Special-Purpose Processors Perform Well?
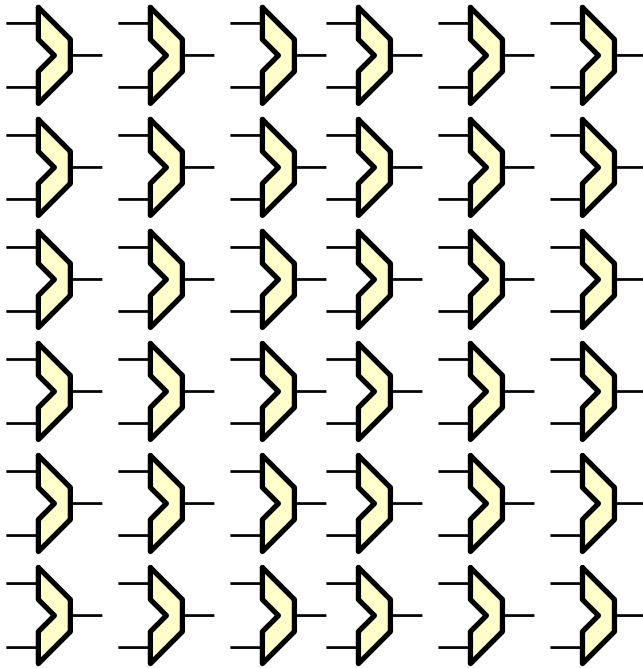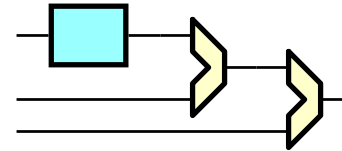
Lots (100s) of ALUs   Fed by dedicated wires/memories

Care and Feeding of ALUs

Instruction
Bandwidth
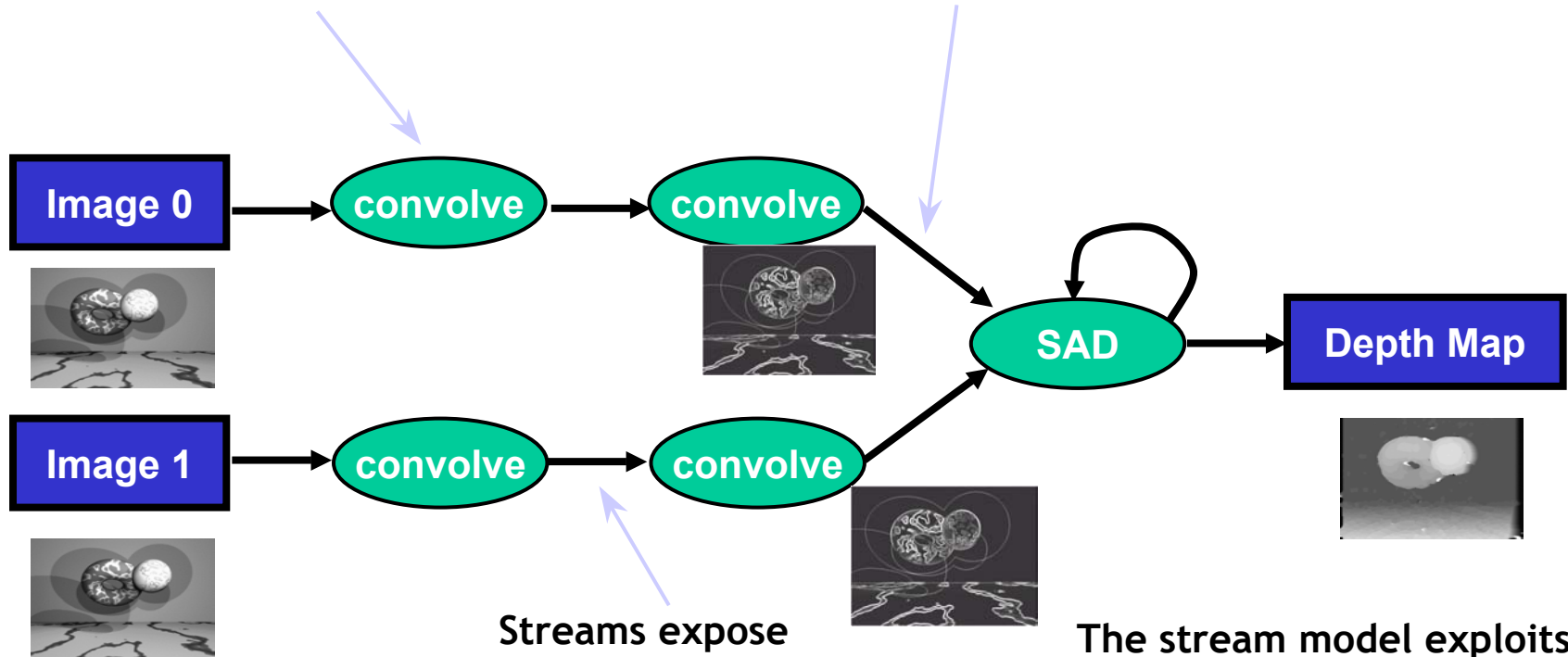
Data
Bandwidth

IP

Instr.
Cache

IR

Regs

'Feeding' Structure Dwarfs ALU

Stream Programs Expose Locality and Concurrency

**Kernels exploit both instruction (ILP) and data (SIMD) level parallelism.**

Kernels can be partitioned across chips to exploit task parallelism.

Streams expose producer-consumer locality.

The stream model exploits parallelism without the complexity of traditional parallel programming.

A Bandwidth Hierarchy exploits locality and concurrency

2GB/s   **32GB/s**   **544GB/s**
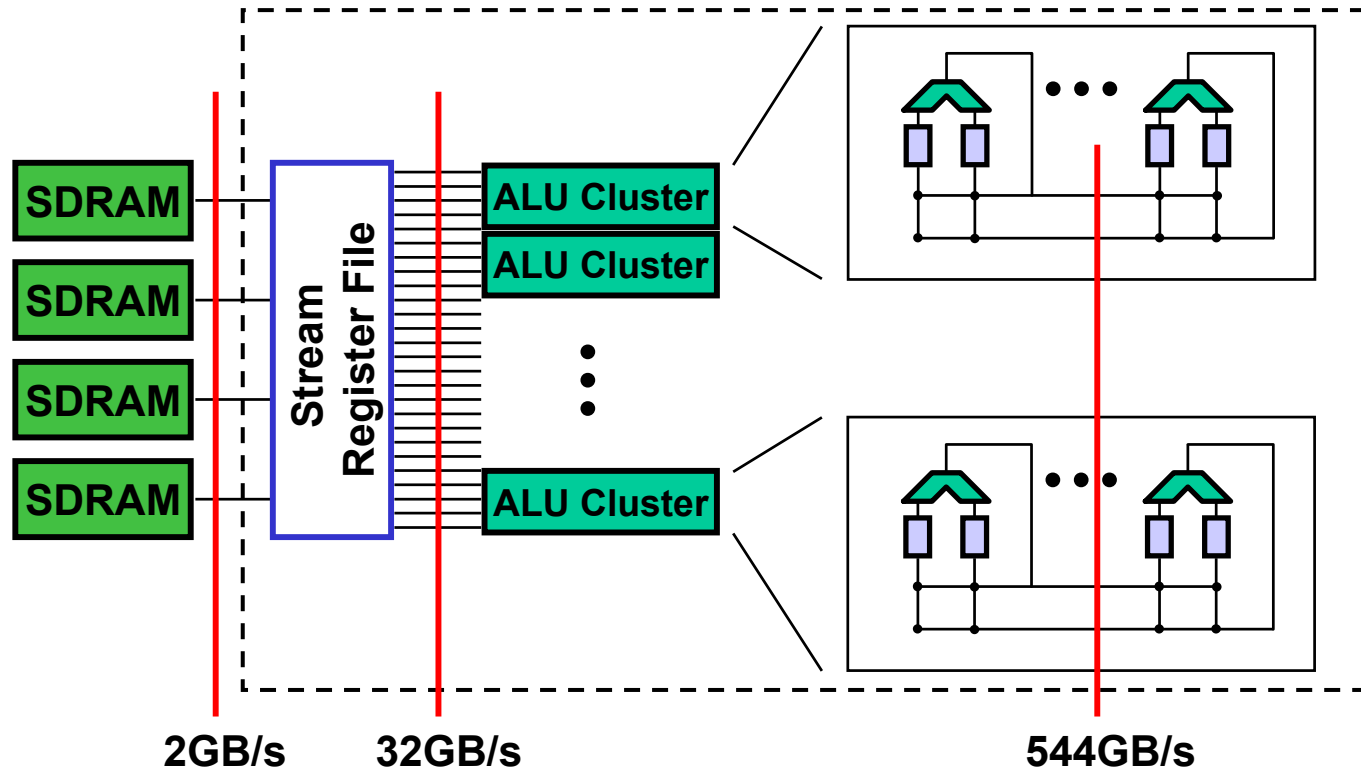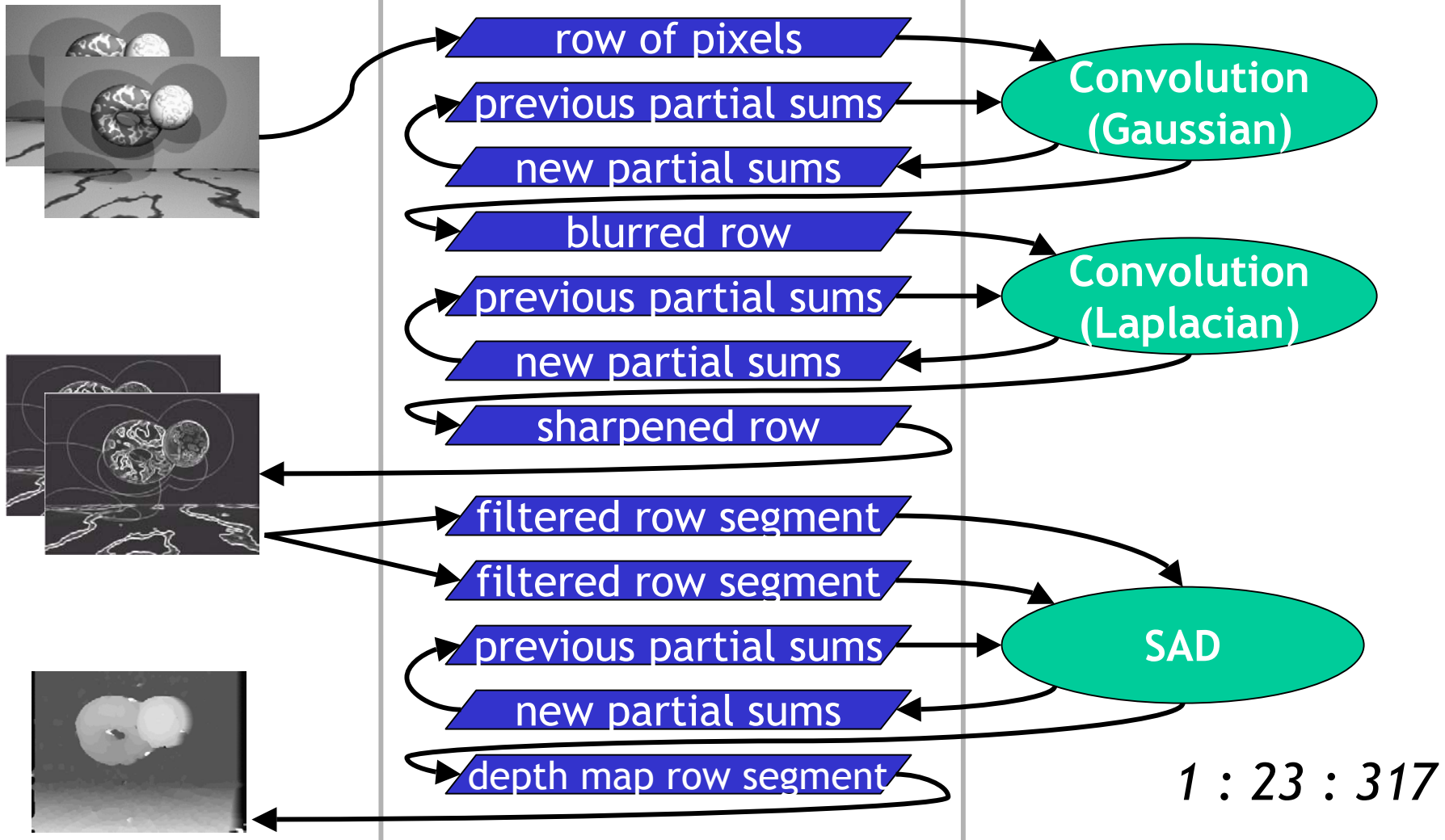
- VLIW clusters with shared control
- 41.2 32-bit floating-point operations per word of memory BW

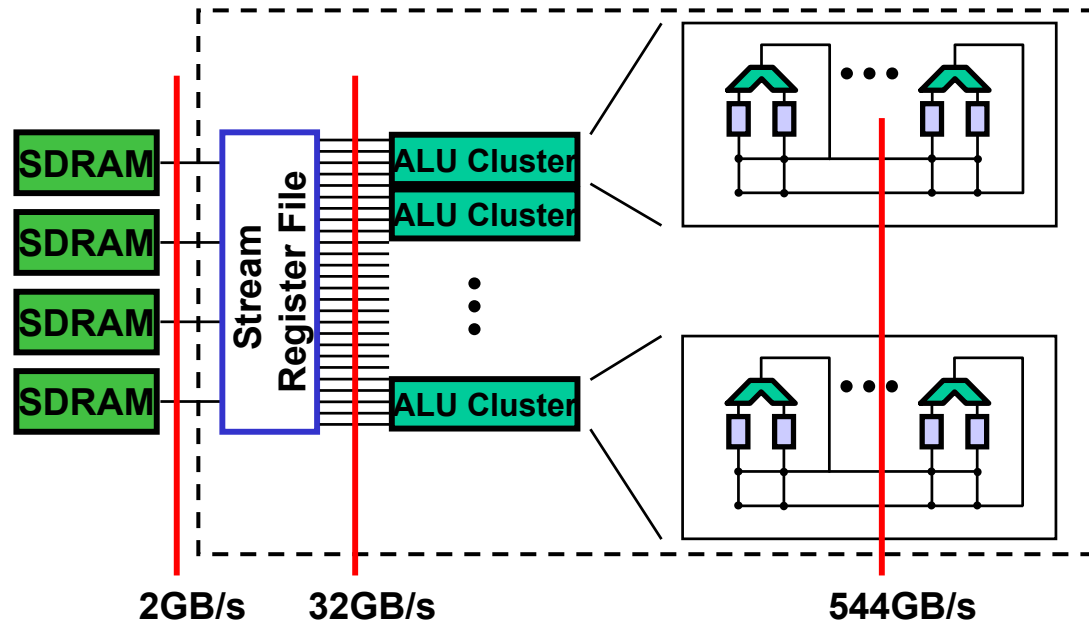# Producer-Consumer Locality in the Depth Extractor

Memory/Global Data   SRF/Streams   Clusters/Kernels

row of pixels

previous partial sums

new partial sums

Convolution (Gaussian)

blurred row

previous partial sums

new partial sums

Convolution (Laplacian)

sharpened row

filtered row segment

filtered row segment

previous partial sums

new partial sums

SAD

depth map row segment

1 : 23 : 317

A Bandwidth Hierarchy exploits kernel and producer-consumer locality

2GB/s   **32GB/s**   **544GB/s**

|	Memory BW	Global RF BW	Local RF BW
Depth Extractor	0.80 GB/s	18.45 GB/s	210.85 GB/s
MPEG Encoder	0.47 GB/s	2.46 GB/s	121.05 GB/s
Polygon Rendering	0.78 GB/s	4.06 GB/s	102.46 GB/s
QR Decomposition	0.46 GB/s	3.67 GB/s	234.57 GB/s

Bandwidth Demand of Applications
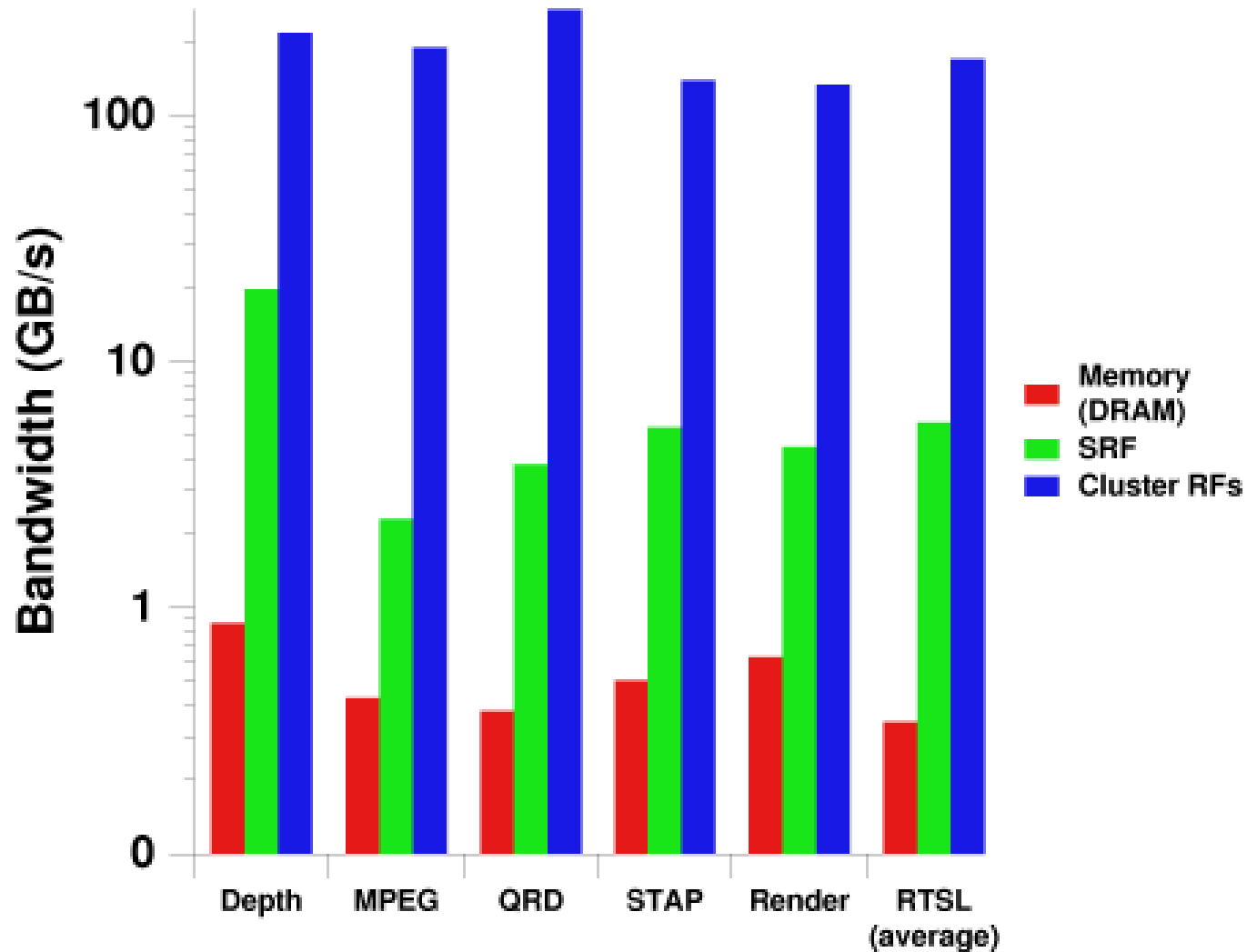
Local registers increase effective size and bandwidth of SRF

- ~90% of live variables are captured in local registers
- Only 10% of live variables need be stored in stream register file
- Fixed-size SRF is effectively 10x the size of a VRF that must hold all live variables
- Bandwidth into FPUs is 10x the SRF bandwidth

Cluster Occupancy > 80%
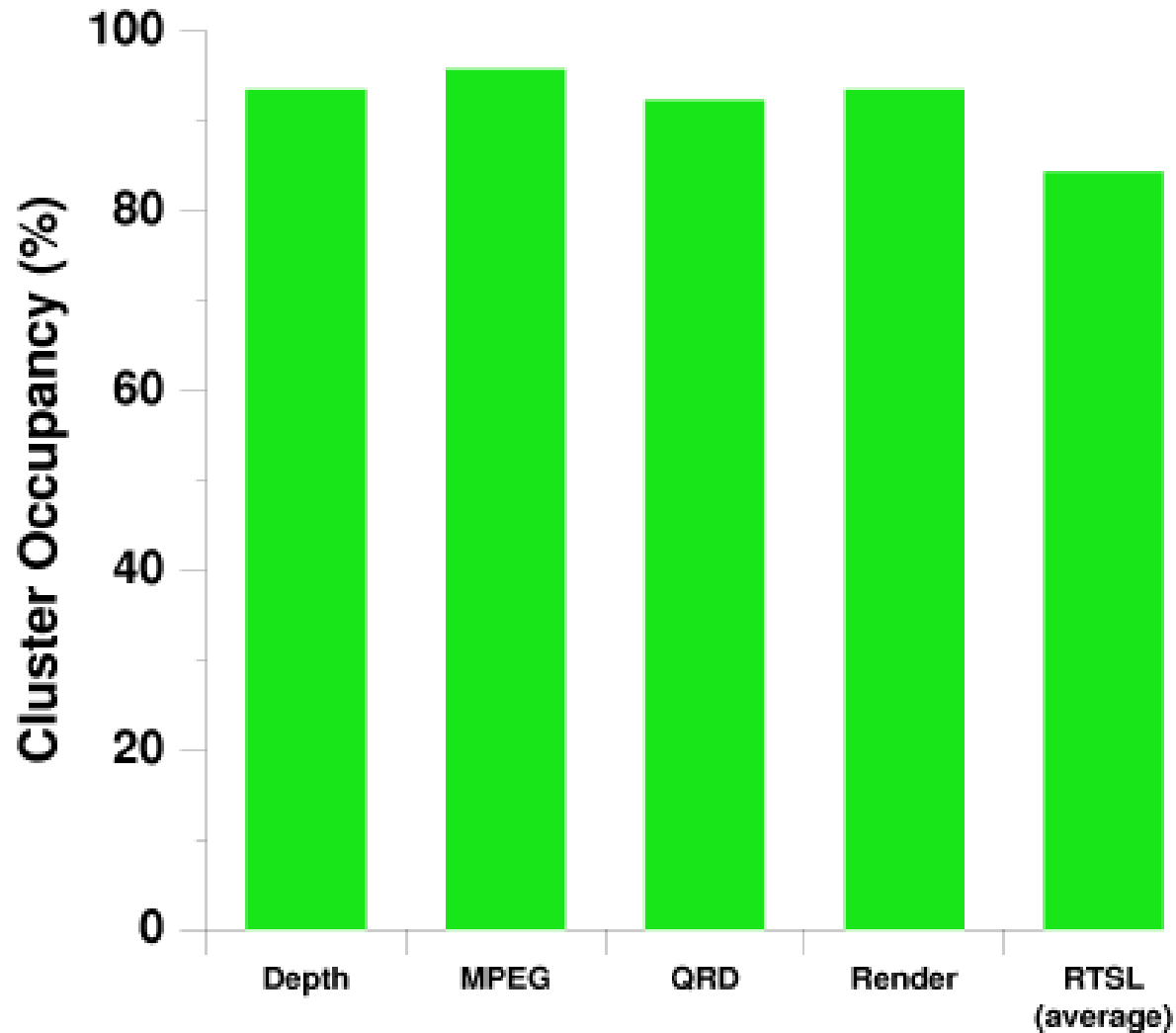
Performance demonstrated on signal and image processing
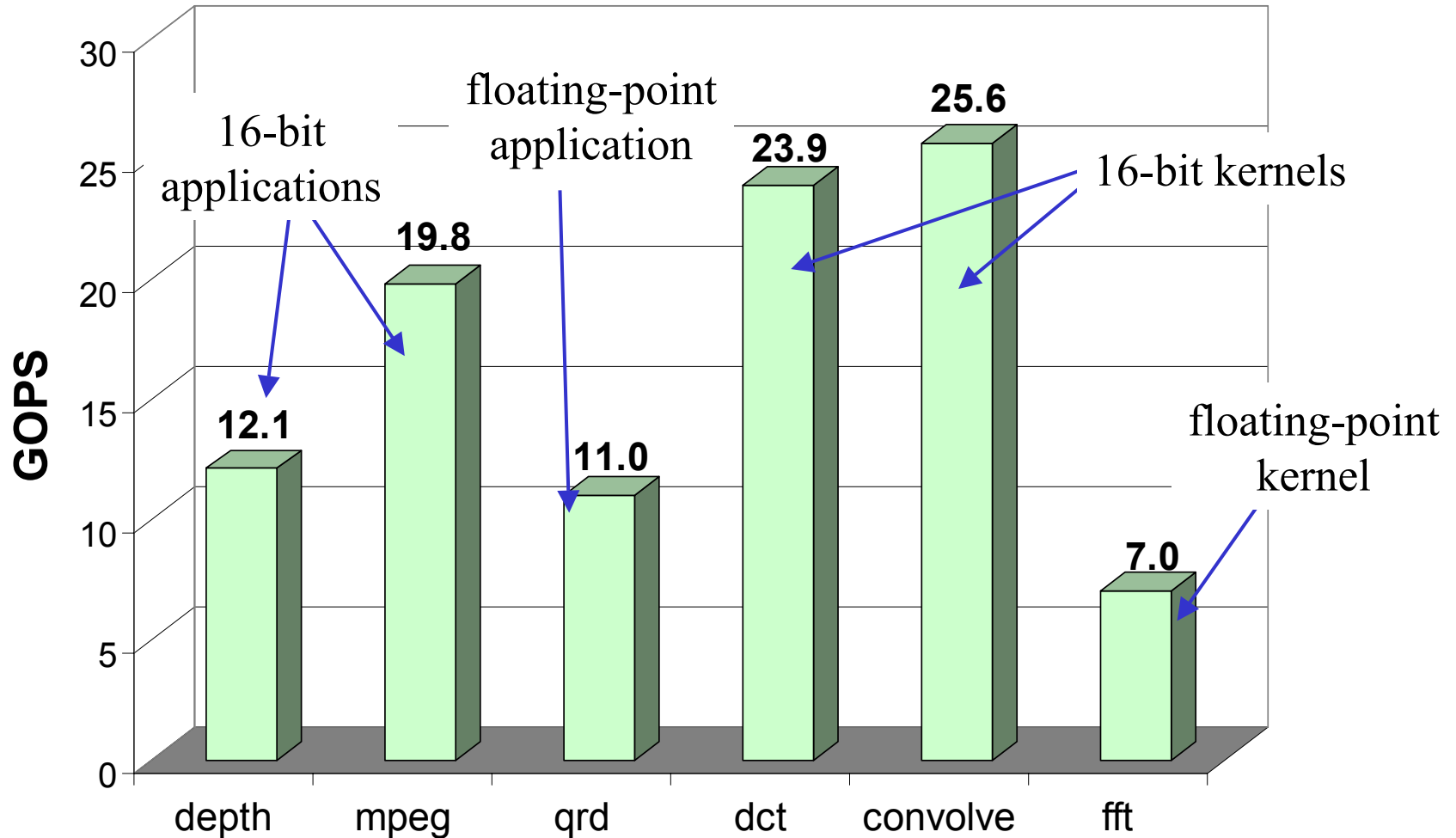
Prototype

- Prototype of Imagine architecture
  - Proof-of-concept 2.56cm^2 die in 0.18um TI process, 21M transistors
  - Collaboration with TI ASIC
  - Runs all benchmarks at 240MHz

- Dual-Imagine development board
  - Platform for rapid application development
  - Test & debug building blocks of a 64-node system
  - Collaboration with ISI-East
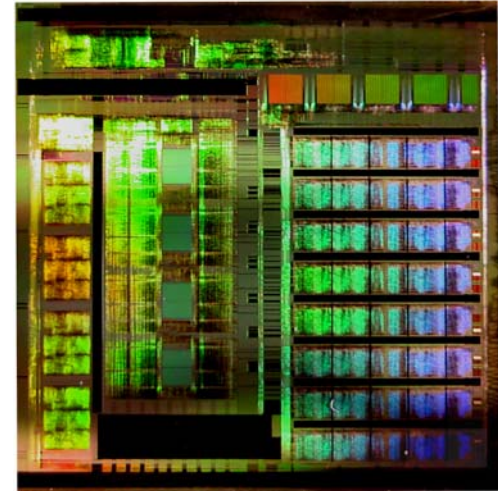
Imagine is programmed in "C" at two levels

- ## Streams:
  - Sequences of records
- ## Kernels:
  - Functions that operate on streams
  - Written in KernelC
  - Compiled by kernel scheduler
- ## Stream program:
  - Defines streams, control- and and data-flow between kernels
  - Written in StreamC and C++
  - Compiled by stream compiler

Span Convert

Rasterize
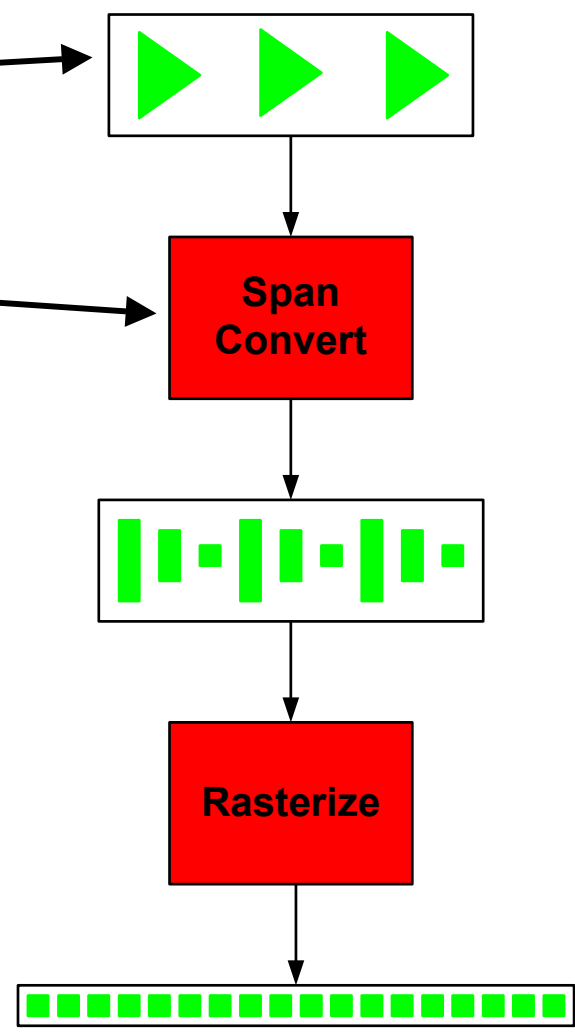
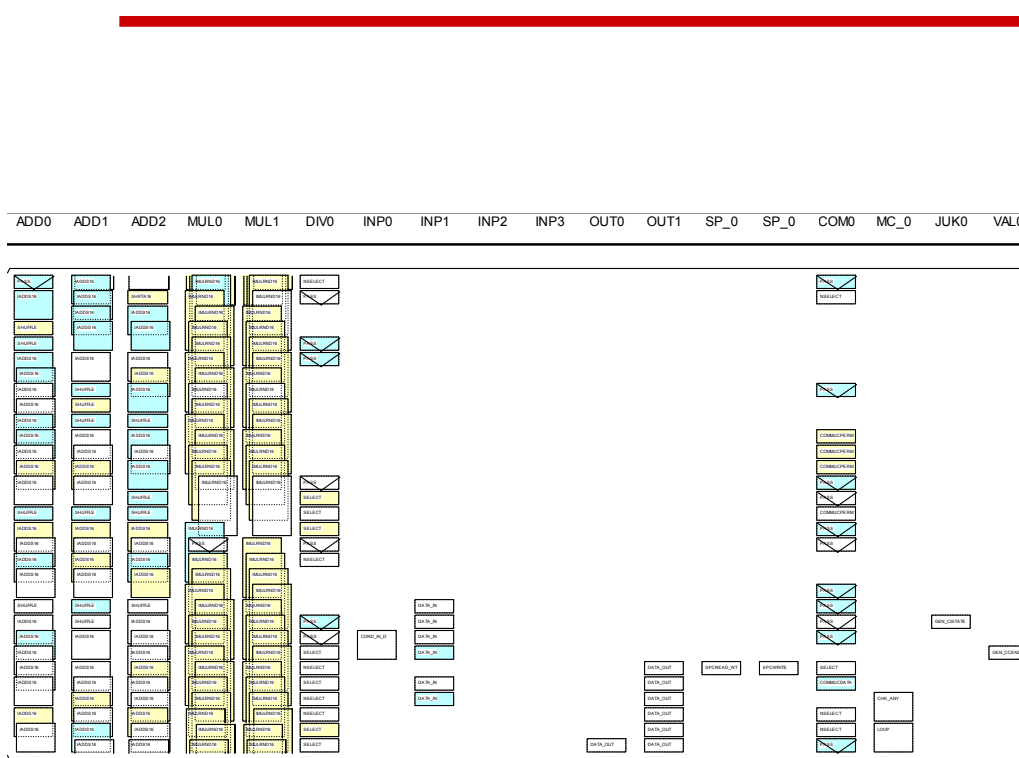# Simple example

- ## StreamC:

```
void main() {
  Stream<int> a(256);
  Stream<int> b(256);
  Stream<int> c(256);
  Stream<int> d(1024);
  ...
  example1(a, b, c);
  example2(c, d);
  ...
}
```

- ## KernelC:

```
KERNEL example1(
  istream<int> a,
  istream<int> b,
  ostream<int> c)
{
  loop_stream(a) {
    int ai, bi, ci;
    a >> ai;
    b >> bi;
    ci = ai * 2 + bi * 3;
    c << ci;
  }
}
```
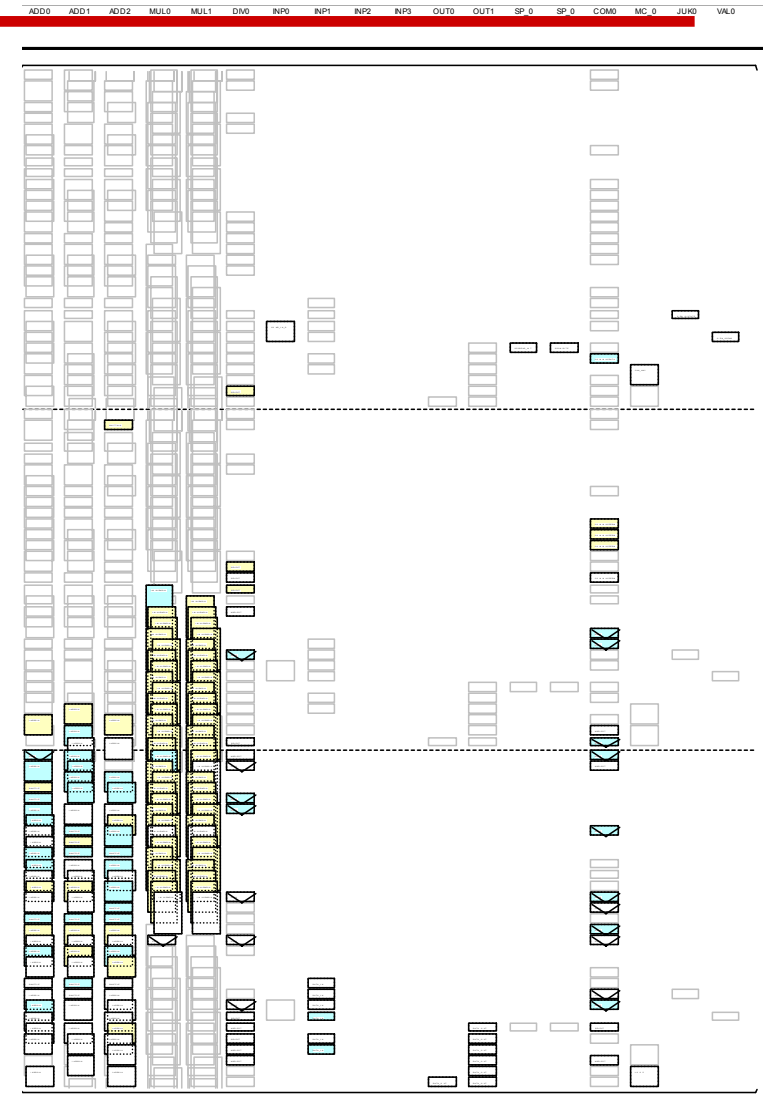
Communication scheduling achieves near optimum kernel performance

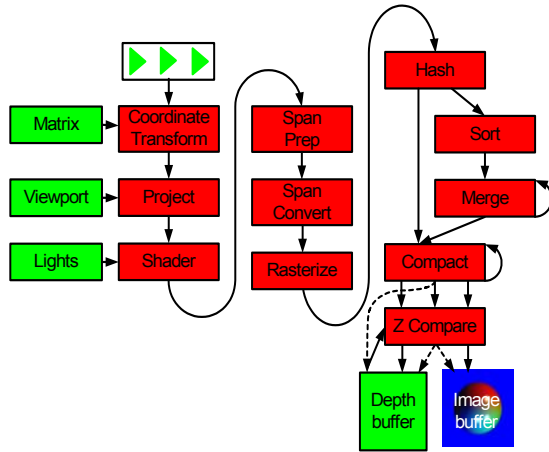7x7 convolution kernel from depth extraction application

(Above) Single iteration schedule

(Right) Software pipelining shown

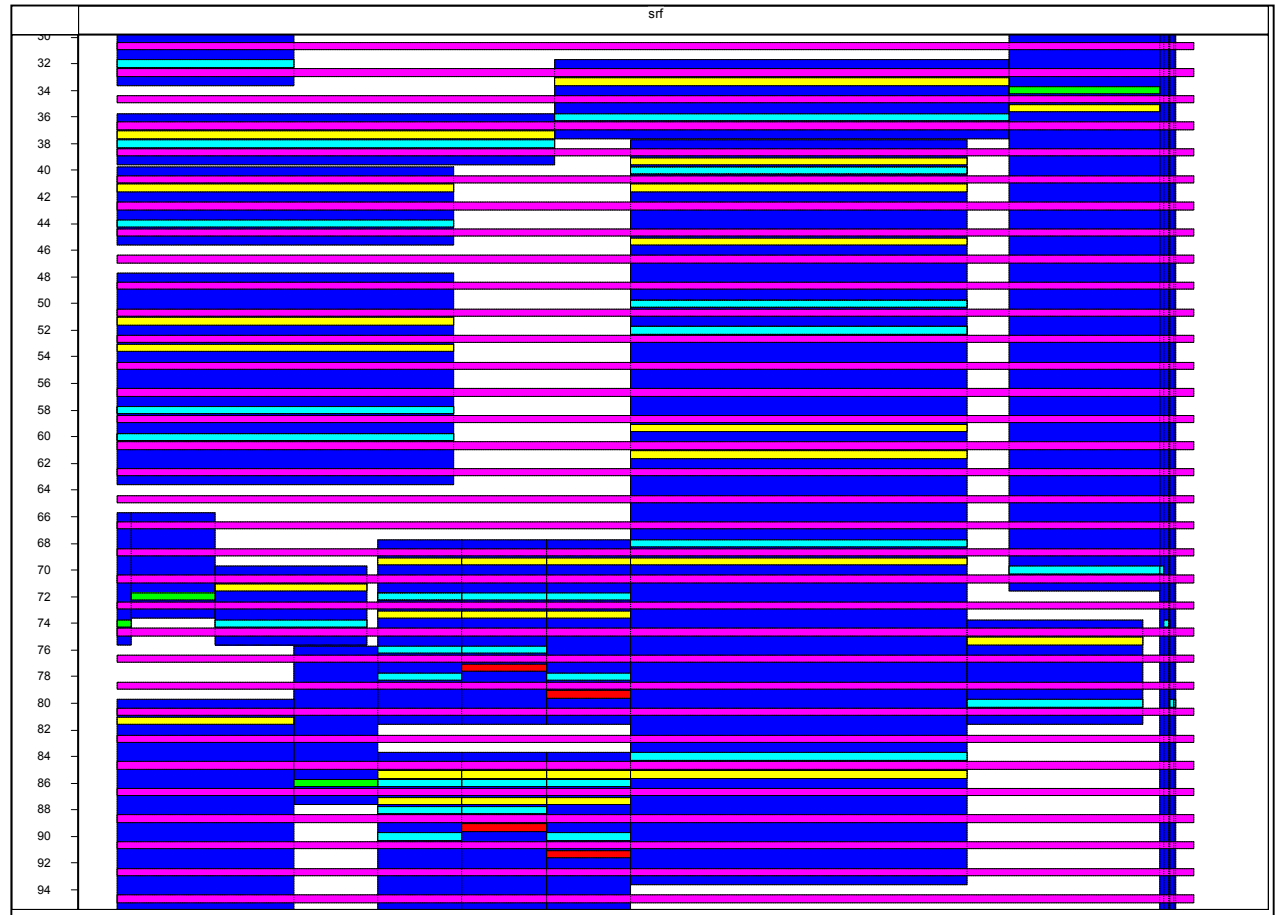# Stream scheduling reduces bandwidth demand by up to 12:1 compared to caching

- Stream program
- SRF allocation

Open GL graphics pipeline

Current DSP programmers attempt to stage data in this manner by hand

We have developed...

- A *stream architecture* that exploits locality and concurrency
  - Keeps 99% of the data accesses on chip
  - Aligned accesses to SRF
  - Enables efficient use of large numbers (100s) of ALUs
- Imagine: a prototype *stream processor* that demonstrates the efficiency of stream architecture
  - Working in the lab at 240MHz
  - 9.6GFLOPS, 19.2GOPS, 6W
  - Programmed in "C"
  - Sustains ~5GOPS/W at 1.2V (200pJ/OP)
- and demonstrated image-processing, signal processing, and graphics applications on the Imagine stream processor

Stream processing can be applied to scientific computing

- ## Extensions to architecture
  - 64b floating point – 100GFLOPS/chip
  - Support 2-D, 3-D, and irregular data structures
    - Stream cache
    - Indexable SRF

- ## Estimates suggest we can achieve
  - <$20/GFLOPS
  - <$10/M-GUPS
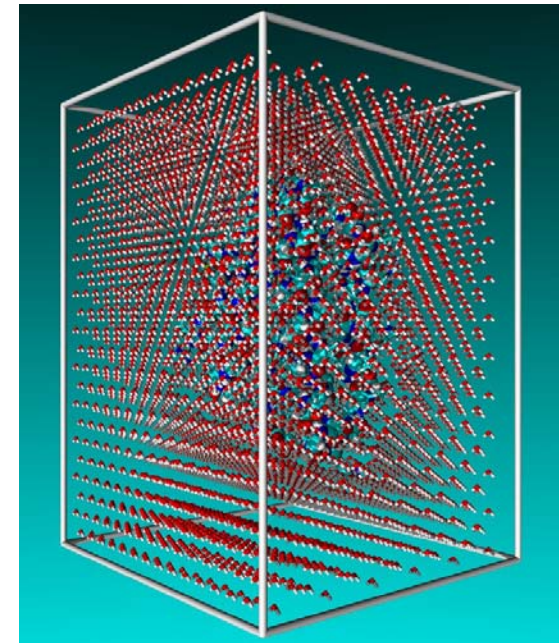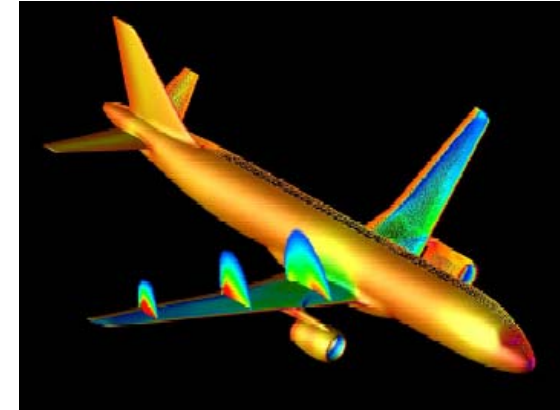
Conclusion

- Streams expose locality and concurrency
  - Concurrency across stream elements
  - Producer/consumer locality
  - Enables compiler optimization at a larger scale than scalar processing
- A stream architecture exploits this to achieve high arithmetic intensity (arithmetic rate/BW)
  - Keeps most (>90%) of data operations local (544GB/s, 10pJ) with low overhead
  - Keeps almost all (>99%) of data operations on chip (32GB/s, 100pJ)
- The Imagine processor demonstrates the advantages of streaming for image and signal processing
  - 9.6GFLOPs, 19.2GOPs, 6W - measured
- Stream processing is applicable to a wide range of applications
  - Scientific computing
  - Packet processing